Archie

Alan Emtage, 8313440

May 2, 1991

Contents

Chapter 1

Introduction

1.1 The Problem

One of the most fundemental changes to have occurred over the past decade in the way humans utilize computers has been in communications. In 1980 the most advanced computer network devoted to research and education was the ARPAnet, originally set up and administered by the Advanced Research Projects Agency of the Department of Defence in the United States. This network consisted primarily of a few thousand hosts connected via communications lines running at speeds of a few thousand bits per second. In the space of a decade demand and growth of the network (now called the Internet) has been so rapid that over 320,000 hosts have access to its resources and the main centers of operation can commicate over links operating at over 4 million bits per second. It is envisioned that by the end of the decade Gigabit (trillions bits/sec) speeds will be in place.

One may ask what is all this capacity being used for. The speed of today's networks allows the realization of applications that is previous years would not have been feasible, or even possible. For example, interactive X window graphical interfaces where data being generated and modeled at a supercomputer in Illinois and being displayed in California would not have been possible until the past couple of years. Similarly, USENET, the Internet bulletin board service which was initially intended as a small application serving about 20 sites now serves over 250,000 sites and there is over 10Mb/day of traffic associated with this service alone.

Some of the information being generated and passed over these networks has long term utility. For example, every year thousands of lines of program code, created by users of the Internet, are released into the public domain (or made freely available) so that anyone wishing to use and modify it for their own purposes may do so. Some of the most well-used software packages on the Internet fall into this category: X Windows from MIT, TEX from Stanford and the GNU Project are examples of these. Documentation, educational material and datasets are also widely distributed on the network. As a direct result of this, certain sites across the Internet have seen fit to make some of their local resources, in the form of disk space, available to the network at large by storing these various forms of information and allowing access to other, non-local users. These sites are known as **Anonymous FTP** or **archive** sites. FTP, the File Transfer Protocol is the network protocol specifically designed for the transfer of files accross the Internet. Normally, in order to access the files on another host, one must have login priviledges on that host. However, many FTP implementations make it possible for the system administrators to set up an **anonymous** account requiring no password, which allows any user on the Internet to access a specific subset of the files on their system.

However until the past 6 months or so, a major problem has existed in managing all this information: there was no central repository that one could query to locate any specific software package or documentation. In fact, there were special-interest groups on the USENET service dedicated for questions like "Where can I find program X ?" and "Which site has the latest version of the Y document ?". In September of 1990 this situation changed with the introduction of archie, the first Internet archive server-server.

1.2 What is archie ?

archie is a database system which retrieves and maintains the file directory listings of several hundred archive sites across the Internet. Users of the Internet may log onto a host running the archie system and query the database as to the location, modification times and size of any program or document that they may be searching for, stored on an anonymous FTP site somewhere on the network. Alternatively archie provides an email interface to the database, which allows those users not directly connected to the network to contact it.

The archie system actually consists of a number of distinct components, which perform such things as the retrieval of the site listings, the updates to the database and the interactive and electronic mail interfaces.

Besides storing site listings, archie also maintains a text database known as the Software Description Database, whose purpose is to provide users of the network with a short description of the thousands of various software packages and documents available on the anonymous FTP sites.

For a history of the archie project see Appendix 1.

1.3 How to use archie ?

The only host on the Internet currently running as an archie server is `quiche.cs.mcgill.ca` (132.206.2.3). In order to use the service, one uses the Internet terminal emulator *telnet*(1). At the login prompt, type "archie". No password is required. The user interface to archie will be described in greater detail later.

1.4 Contributors

The various parts of archie have been written by the following people.

- I have been responsible for overall system design as well as the design and coding of the database and its associated routines. Day to day database maintenance (updates) is also carried out by myself.
- Bill Heelan (wheelan@cs.mcgill.ca) wrote the user interface and has had significant input into the design of the system.
- Mike Parker (mouse@larry.mcrcim.mcgill.ca) wrote the initial email interface, based on the KISS server.
- Peter Deutsch (peterd@cc.mcgill.ca) provides system resources and ongoing input into the user interface design.
- Edward Vielmetti (emv@msen.com), moderator of the USENET newsgroup comp.archives has provided a wealth of suggestions and constructive critisism as well as pointing out new sources of information for archie.
- The code for the archie help facility is derived from the **gnuplot** utility written by Thomas Williams, Pixar Corporation, (pixar!info-gnuplot@sun.com).
- Code for the SMTP section of the archie interactive "help" command comes from the **c-client** library by Mark Crispin (mrc@wsmr-simtel20.army.mil).

Chapter 2

The User Interface

As mentioned before archie has both an interactive and an email user interface. Every utility under UNIX usually has a manual page describing its use. What follows is the manual entry for archie.

p

Chapter 3

The Design

The first archie database consisted of the raw directory listings from the anonymous ftp sites and archie was a simple login interface to *egrep*(1). However, once it was decided that archie was to be a system in its own right, a database structure had to be designed. This structure had to fulfill the following requirements:

- Easily provide the functionality of the two existing commands **site** and **prog**. That is, allow a specific site to be listed, and allow the database to be searched for a specific string and the matches shown.
- Be flexible in the type of search that could be performed. Initially, regular expressions were used since they provide the most general means for string searches. To a lesser extent (since *egrep* was initially used) this was what archie users had become accustomed to. However the database structure had to provide the flexibility for other (yet unspecified) searches to be performed.
- Speed. Although always a concern, archie subscribes to the maxim "Get it working first, then make it fast". Since no service akin to archie existed on the net, we were interested in getting a prototype up and running as soon as possible, since refinements could be added later.
- Space. The raw listings contained large amounts of redundant information and padding (white space). The same permission field string can be repeated hundred or thousands of times through the listing, similarly for group, owner and link fields. It was thought that considerable compression could be obtained in the new structure. [An assumption that turned out to be wrong for the most part].

The main concept behind the design of archie was that of relational databases. That is, try wherever possible to store a piece of information only once in the database. When needed this information could can be "pointed at". This process would usually show a reduction in space, with a possible tradeoff in running time depending on how expensive memory references were on the architecture on which you were running.

In a system such as we describe here, it is important for the database designer to realize that the interactive use of the facility be made as efficient as possible and that since updates are done to be done in batch mode at off-peak hours, as much time as needed should (and can) be spent preprocessing the information.

3.1 The Strings

Although allowing the user to obtain individual site listings was, and still is an integral part of archie's functionality, it was assumed (and soon confirmed) that the most common operation to be performed on the database would be string searches. Thus designs that enhanced search times were favoured. Drawing on the relational database concept, it was decided that only one occurrence of any given string would occur in the database and initial ideas moved towards the concept of a tree-like structure for their storage. However, it was soon discovered that no well-known tree structure provided the flexibility required: not only were substring searches required but for the more advanced user, regular expressions provided a powerful method for finding the software being sought. In addition the ability to provide **exact match** searches, (wherein the user knows the exact string she or he wants to find), at a very low cost seemed an attractive feature. The only structure that provided all of the criteria laid out above was essentially a flat file (with pointer information to the other database files). We have found this to be an adequate solution and with a good search algorithm, good to excellent performance may be obtained.

It should be noted that it was decided that at the search level, there would be no distinction made between files and directories in the system. Thus a search for a specific string may reveal a file *or* directory with that string (how the two relate depends on the kind of search being performed).

3.2 The Addresses

Another one of the early design decisions was the use of the binary representation of an IP address when referring to a particular site's "name". Benefits here were both in speed and time: the UNIX networking code works with this format (as a 32-bit quantity); the representation was certainly more compact than either of the alternatives: the *dotted decimal notation* (*eg* 132.206.3.3) or the *fully qualified domain name* (*eg* quiche.cs.mcgill.ca).

In one of the first prototypes of archie, it was discovered that using the Domain Name System to resolve the IP addresses stored in the database (for human consumption) was unacceptable in terms of time spent waiting for a response from the network. Thus it was decided that a DNS cache would be implemented which would perform the lookups for the subset of Internet sites stored in the database at any given time. Special consideration has to be given to IP addresses changing and how to obtain the primary names and addresses of each site but experience has shown that the current implementation work reasonably well.

3.3 Tying it together

The final decision had to do with linking the files with the same name on different sites together. The simplest and most efficient way of doing this is the linked list. The string is associated with a pointer to the head of a linked list whose components consist of pointers to the file specific information. This way, by simply running down the list, all files with a common name are immediately associated.

Chapter 4

The Database

Having seen the criteria and design decisions, in this chapter we will describe the final layout of the database.In the following discussion, the anonymous ftp listing shown in Figure 1 will be used to illustrate the various database elements.

The final database structure take form consisted of 4 major components (See Figure 2).

It was decided that although the code for archie itself should be as portable as possible, the database need not be so. That is, it is not expected that the database files should need to be transferred between different hardware architectures. As a result, the various database structures are written to disk as they are stored in core. This facilitates the quick reading and writing of database records to and from disk without having to translate between an external (disk) and internal (core) format. This assumption may very well be wrong and thus this may need to be changed in the future.

Implementation note: The arrangement of the various fields in the structures described below was chosen so that the amount of padding placed within it was at a minimum and so the record was as compact as possible for a Sun 4/280 (Sparc architecture). If your compiler/hardware configuration pads its structures differently, they may be modified to suit. No other code modification should be necessary.

```
total 3
dr-xr-xr-x  2 0         staff         512 Jul 10  1990 bin
dr-xr-xr-x  2 0         staff         512 Jul 10  1990 etc
drwxrwxr-x  4 ftp       staff         512 Feb 12 03:27 pub

bin:
total 44
---x--x--x  1 0         staff       45056 Jul 10  1990 ls

etc:
total 2
-r--r--r--  1 0         staff          40 Jul 10  1990 group
-r--r--r--  1 0         staff          39 Jul 10  1990 passwd

pub:
total 58
-rw-r--r--  1 269       staff       21341 Jan 16 00:03 ftp+readline_patch.tar.Z
-rw-r--r--  1 269       staff       35709 Jan  5 23:03 man-1.0.tar.Z
drwxr-xr-x  4 269       staff         512 Feb 12 03:27 tex
drwxrwxr-x  3 ftp       staff         512 Dec 19 23:52 users

pub/tex:
total 2
drwxr-xr-x  2 269       staff         512 Feb 12 03:50 fonts
drwxr-xr-x  2 269       staff         512 Feb 12 03:27 macros

pub/tex/fonts:
total 46
-rw-r--r--  1 269       staff         536 Feb 12 03:50 README
-rw-r--r--  1 269       staff       18724 Feb 12 03:27 utseal.240pk
-rw-r--r--  1 269       staff       25096 Feb 12 03:27 utseal.300pk
-rw-r--r--  1 269       staff         164 Feb 12 03:27 utseal.tfm

pub/tex/macros:
total 0

pub/users:
total 1
drwxr-xr-x  2 269       staff         512 Dec 20 00:03 jwe

pub/users/jwe:
total 174
-rw-r--r--  1 269       staff         338 Dec 20 00:03 README
-rw-r--r--  1 269       staff      103760 Dec 19 23:52 articles.bib
-rw-r--r--  1 269       staff       31392 Dec 19 23:52 books.bib
-rw-r--r--  1 269       staff       21228 Dec 19 23:52 proceedings.bib
-rw-r--r--  1 269       staff        8779 Dec 19 23:52 unpub.bib
```

Figure 1

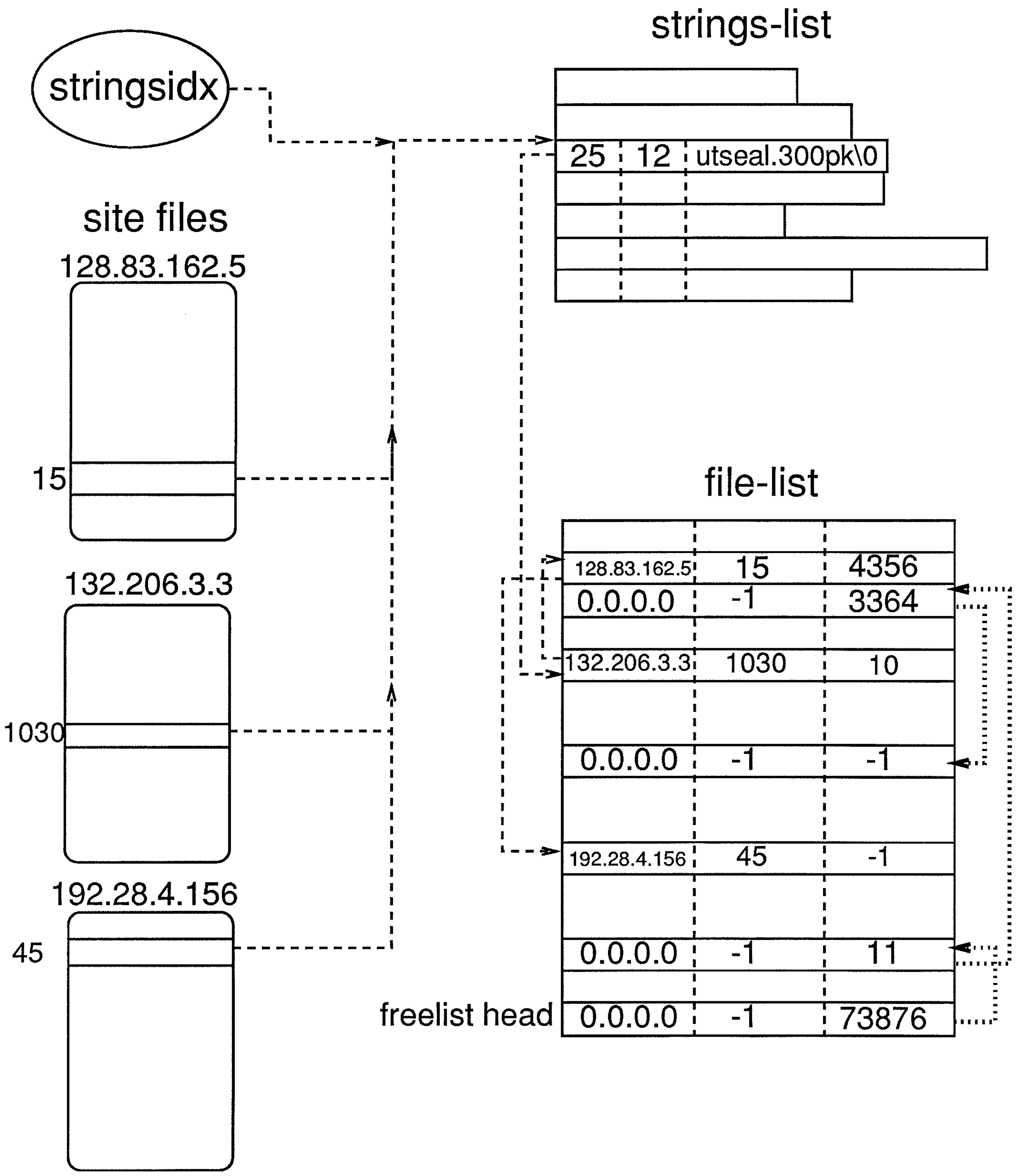

strings-list
stringsidx
25
12
utseal.300pk\0
site files
128.83.162.5
15
132.206.3.3
1030
192.28.4.156
45
file-list
128.83.162.5 15 4356
0.0.0.0 -1 3364
132.206.3.3 1030 10
0.0.0.0 -1 -1
192.28.4.156 45 -1
0.0.0.0 -1 11
freelist head
0.0.0.0 -1 73876

Figure 2

4.1 The Database Files

- The **strings-list** table. Drawing on the idea of relational databases, each unique string (directory or file name) is stored once in the database. Thus although almost every site will have a "bin" subdirectory, the string "bin" is only stored in one place in the archie structure. In the current structure the string is stored NULL terminated. Associated with each string is its length and an index into the **file-list** (described below). Note that the string length stored does not include the NULL character at the end. Thus the table consists of variable length records of index, length and string.

  This lead to the following C structure for the header to each string:

```
typedef struct{
        long filet_index;
        short str_len;
} strings_header;
```

  The entries in this file are never deleted. If no more files in the database are found to have this name, then a sentinel value of -1 is placed in the `filet_index` field to mark it as such.

  As mentioned before, this structure is written to disk from core. There is no translation into ASCII so the values on disk are in binary format.

- The site tables. For every site stored in the database, a **site** file is created. The name of this file is the ASCII "dotted decimal" representation of the IP address for this site. The site file consists of records from the following structure:

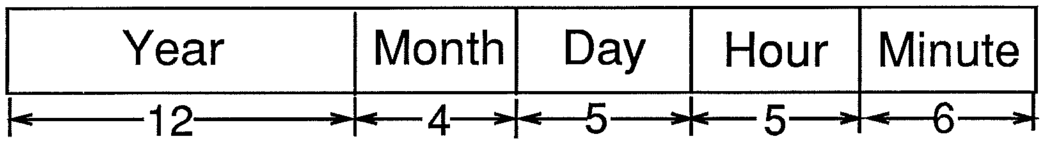
```
struct site_entry{
       size_type size;                   /* Size of file                  */
       parenti_type parent_ind;          /* Record number of parent       */
       fchildi_type first_child_ind;     /* Record number of first child */
       db_date mod_time;                 /* Modification time             */
       union{
            stringsi_type strings_ind;   /* Index of name                 */
            struct in_addr ipaddress;    /* or site IP addr (root rec)    */
       }in_or_addr;
       perms_type perms;                 /* File permissions              */
       char dir_or_f;                    /* Directory or file flag        */

};
```

  See Figure 3. Each record corresponds to a file or directory in the original listing. Note that the owner, group and link information from the original has been discarded. It was considered that this information was of negligible use in the context of archie.

  The site table is constructed to mirror the standard UNIX *ls*(1) listing style. That is, a depth first, in-order search of the directory tree. Thus the root is listed first, followed by the first directory in it. If this subdirectory itself contains directories they too are listed in order etc. Only when all subdirectories have been listed does the second root subdirectory get listed and so on.

  - The `size` field is an unsigned 32 bit field `size_type` containing the size of file in the listing.
  - `parent_ind` Contains the record number of the parent of the current file (or directory). 32 bit quantity `parenti_type`
  - `first_child_ind` Contains the record number of the first child if the current record is a directory, otherwise the value is undefined. Since empty directories are not entered into the database, no sentinel value for this case is required. 32 bit quantity `fchildi_type`.
  - The modification date is stored in a 32 bit quantity (`mod_time`, type `dbdate`) in "binary coded decimal" format. Sorting of entries by date works correctly. A speed bottleneck was found when the standard UNIX `time_t` format (number of seconds since 00:00 Jan 1 1970 GMT) was used since various multiplications and

divisions have to be performed in order to convert this to an standard date ASCII string and this slowed performance measurably. The current format was found to be ideal for our needs.



- The file permissions (**perms**, 16 bit quantity **perms_type**) are stored in bitwise form corresponding to the lower 9 permission bits in the standard UNIX form. See *chmod*(1).
- The **dir_or_f** field need not be a full character in length and in fact could be moved into the **perms** field which is defined as a 16 bit quantity and is only using 9 bits. It is expected that this will occur in the near future. **dir_or_f** has the value 'T' if it is a directory, 'F' otherwise. It is an 8 bit quantity.
- The **strings_ind** field contains a byte offset into the strings-list table marking the start of the **strings-list** record for the name of the current file. It is of type **stringsi_type** (32 bits).

4.1.1 The Root Record

The first record (number 0) has special significance in the site file. It is known as the **root record**. The **size** and **perms** entry for the root record are undefined. In this record, The union which normally contains the offset into the strings table **in_or_addr** contains instead the IP address of the the site in network byte order as a 32 bit quantity (type **struct in_addr**). The **mod_time** field contains the date of last update for that site.

```
Record Size          Parent  First Child  Modification     Strings   Perms Type
  no      index     Index       date    index

 0: -134222152        0         0      2087863055      4160747824      0   T
 1:        512        0         4      2087081983              17    365   T
 2:        512        0         5      2087081983              41    365   T
 3:        512        0         7      2087805147              64    509   T
 4:      45056        1         0      2087081983             231     73   F
 5:         40        2         0      2087081983             253    292   F
 6:         39        2         0      2087081983             267    292   F
 7:      21341        3         0      2087747587         7166632    420   F
 8:      35709        3         0      2087726531         7166665    420   F
 9:        512        3        11      2087805147            9641    493   T
10:        512        3        17      2087427572          655649    509   T
11:        512        9        13      2087805170           10943    493   T
12:        512        9         0      2087805147          658400    493   T
13:        536       11         0      2087805170             403    420   F
14:      18724       11         0      2087805147         7166687    420   F
15:      25096       11         0      2087805147         7166708    420   F
16:        164       11         0      2087805147         7166729    420   F
17:        512       10        18      2087428099         7166748    493   T
18:        338       17         0      2087428099             403    420   F
19:     103760       17         0      2087427572         7166760    420   F
20:      31392       17         0      2087427572         6651585    420   F
21:      21228       17         0      2087427572         7166781    420   F
22:       8779       17         0      2087427572         7166805    420   F
```

Figure 3

The **strings-ind** entry for the root record (record 0) is the site IP address as a 32-bit value. This number corresponds to 128.83.162.5 in “dotted decimal” notation.

- The **file-list** table allows all entries (files or directories) with the same name in the database to be related. This is done by forming linked lists. The head of the linked list is pointed to by the corresponding entry in the **strings-list**. The C structure describing the records in the file list is

```
struct file_entry{
        struct in_addr site_addr;     /* stored as 32 bit IP address    */
        sitei_type site_ind;          /* record index within the file   */
        filei_type next_ind;          /* pointer to next entry in chain */

};
```

  - The `site_addr` field is translated into the ASCII "dotted decimal" form of the IP address in order to obtain the corresponding site file name. It is an unsigned 32 bit quantity of type `struct in_addr`.
  - The `site_ind` is an index into the site file. This points to the entry that this file-list record represents. This is a signed 32 bit quantity of type `sitei_type`.
  - The `next_ind` field contains a signed 32 bit value of type `filei_type` and is the number of the next record in the linked list for this `strings-list` entry. The "end of list" is marked by a sentinel value of -1.

4.1.2 The Freelist

In addition to the active records in linked lists that the **file-list** contains at any given time, there are a number of records which have been "deleted". In order to efficiently utilize the space of the **file-list** (which can be very large since it contains an entry for every file in every listing), a linked list of inactive records is maintained. This is known as the **freelist**. The idea is that when a site is deleted from the database (see *Updating the Database*), the **file-list** records associated with it are added to the freelist. Similarly, when a site is added, nodes off the freelist are used in preference to expanding the file. Only when the freelist is empty is the file physically extended. This eliminates the need for having garbage collection in the **file-list**.

A special **freelist head** record is maintained as the last record in the **file-list**. It and all other freelist records have a sentinel value of -1 in the `site_ind` field. The `next_ind` field contains the number of the next record on the freelist. As with active records, a value of -1 marks the end of the list. (See Figure 2).

- The **stringsidx** database is a *ndbm*(3) hashed database. It is used in the entry of items as well as in a fast search method. As with all *ndmb*(3) databases a key/value pair is stored for every item both of which conform to the C structure:

  ```
  typedef struct {
        char *dptr;
        int dsize;
  } datum;
  ```

  In this case, the `dptr` field of the key is composed of the NULL-terminated ASCII string for the file/directory name. That is the entries in the **strings-list** file are mirrored in this database. The `dsize` field is the length of the string including the NULL at the end.

  The "value" stored is a byte offset into the **strings-list** file of the corresponding string. It is a 32 bit binary value.

- In order to have quick conversion between Fully Qualified Domain Names (FQDN) and corresponding IP addresses, it was found in the early stages of archie, that some form of caching was required. Since the IP addresses, and not the FQDN are stored in the database, some form of reverse lookup was needed which didn't require the use of the standard Domain Name Server since this was found to be too slow.

  There are two *ndbm*(3) databases which implement this scheme:

  - **hbyaddr** has as it's key the 32 bit IP address (size is `sizeof(struct in_addr)`. The value component is the primary FQDN for that address in ASCII with null termination (size is `strlen(fqdn) + 1`)
  - **hbyname** is the reverse of above: keyed on (NULL terminated) ASCII FQDN; value is 32bit binary IP address (length `sizeof(struct in_addr)`).

  The maintenance of these databases is described in the section *Updating the Database.*

4.2 Updating the Database

Having seen the structure of the database, the description of how updates are performed becomes rather simple. Note that archie performs the update of every site approximately once a month. This is first done by deleting the current entry in the database and entering the new listings. No attempt at finding the difference between the two listings or any such scheme is used.

4.2.1 Deleting

Deleting is performed by the **delete** program. The algorithm is as follows.

- Skipping the root record, read each record sequentially in the site file for to the site to be deleted.
- For each record, locate the **string-list** entry and check to see that it points to an active **file-list** chain, that is, the value of `filet_index` is not -1.Technically it is an error for the value to be -1, since the database is inconsistent. However, if the delete was previously started and then aborted, this condition may occur. It will be flagged if the verbose (-v) flag is given to **delete**.
- Delete the site FQDN and IP address from the DNS caches.
- Follow the **file-list** linked list until the record corresponding to the current entry being deleted is found, that is until the record with the IP address and site record number of the current site record is located. Perform the standard linked list deletion: have the previous element on the linked list point to the element following the one to be deleted. Note that if the element to be deleted is the first one on the chain, then the **strings-list** record will have to be rewritten to point to the new head of the chain. Also, aborted deletes cause the record on the chain not to be found, since it was previously deleted. This error is flagged if delete is invoked with the verbose (-v) flag.
- Add the deleted record to the freelist at the end, using the standard linked list insertion method.
- Continue until all records in the site file have been processed successfully.
- Unlink the site file.

4.2.2 Entering

The Preprocessor

Before being given to the **enter** program which inserts a site into the database, the raw directory listing is preprocessed in order clean up some of the common errors found in these files. For example, often soft links on the ftp site are not resolved and thus error messages to this effect are generated. However these messages are written to the standard error **not** standard output which is what the directory listing is written to. As a result the error messages are mixed into the directory listing in a pseudo-random fashion. It is easier to deal with this kind of corruption in a UNIX *sed*(1) script than it would be to have the parser recognize it. The script is listed in Appendix B.

The Parser

Currently the only parser module for archie expects as input a *ls*(1) long, recursive directory listing. It then builds an internal representation of this listing from which the database update can proceed. The reason for doing it in this fashion is so that the entire listing can be parsed and any errors in it detected before the database on disk is modified in any way. The internal representation of the listing is a tree which roughly mimics the original directory structure at the ftp site composed of the following C structures.

```
typedef struct node{
        int           size;                   /* size of file            */
        int           numsubdirs;             /* number of subdirectories  */
        int           maxsubdirs;             /* curr max for subdirectories*/
        struct node   *subdirs;               /* list of subdirectories    */
        struct node   *parent;                /* parent pointer            */
        struct node   *children;              /* list of children          */
        struct node   *next_file;             /* list of siblings          */
        boolean       directory;              /* is current file a directory*/
        char          owner[MAX_OWNER_LEN];   /* owner string              */
        char          group[MAX_GROUP_LEN];   /* group string              */
        short         perms;                  /* perms as 9 bit integer    */
        char          name[MAX_NAME_LEN];     /* filename                  */
        int           record_no;              /* record # in site file     */
        datestruct    lacess;                 /* last modification date    */

} node;
```

The tree is built in the directory listing order.

The interface between the parser and routines that build the tree is defined so that parsers for other types of listings (VMS for example) can be easily written and added.

Database Entry

Once the parser has scanned in the listings and built the tree database entry can proceed with the knowledge that the directory listing was free of errors. This is done under the following algorithm.

- Write the root record for the site file, with the appropriate date (taken from the modification time of the site listing), address *etc.*
- Recursively descend the internal tree as *ls*(1) would, that is a combination of depth first and level-order search.
    1. For each node encountered on the tree check to see if the string exists in the database by searching for it in the **stringsidx** hashed database. If it doesn't then add it to the hashed database and the **strings-list** file.
    2. Get the next available **file-list** node. Either by getting it off of the freelist or extending the file.
    3. Add the record to the **file-list** by inserting it at the head of the linked list pointed to by the **strings-list** file and rewrite the **strings-list** record to point to the new head. By this time the position of the record in the site file is known.
    4. Add the site file record for the node being added.
- Add the FQDN and IP address to the DNS caches.

Chapter 5

Summary

It as proven very difficult to get object results on the performance of archie. The primary reason for this is that it runs in a production environment on a host which performs such widely varying activities as subnet gateway and USENET server for the McGill Campus.

Up until this time archie has existed in two versions. The original was written using standard UNIX standard I/O routines (such as *fread*(3) and *fwrite*(3). However since these routines perform internal buffering it was thought that a more efficient method would be to use the shared memory model provided by 4.3 Berkeley UNIX systems (and this in SunOS) via the *mmap*(2) system call. Since the interface routines access the database in read-only mode, it was reasonable to assume that the various processes could share their copies of the database files after having been read into core. Our experience has shown that a significant speedup is attained via this method although much larger amounts of memory are used. archie is currently running on a Sun 4/280 (about 10 Mips) with 32Mb of core and 70Mb of virtual memory. This machine can comfortably support 20 concurrent archie sessions (in addition to its other activities) and often has over 30 running at any one time.

It is hoped that in the future, objective measurements of the performance of archie will be made.

archie has proven to be one of the most popular services on the Internet and it is expected that its phenomenal growth will be sustained for sometime yet to come.

Appendix 1
Archie Past, Present and Future

In order to provide a more complete understanding of the significance of archie, it is necessary to give a rough outline of the history of the project.

About 2 1/2 years ago, I started collecting the directory listings of my favourite archive sites in order to speed up my search for PD software. Being responsible for installing an maintaining network software for the system staff at the School of Computer Science (SOCS), I spent much of my time using FTP to find software considered to be useful by the users of the system. At the time SOCS was the only Internet connected site in Quebec with a communications link running at 9600 baud. We could literally wait hours for a transfer to complete: for example, X11 Release 2 package took about 2 days. It made much more sense to perform and store a recursive directory listing every couple of weeks than to have to list the files ever time we signed on to that anonymous FTP site.

As more archive sites came online, I collected more listings. Finally, my boss Peter Deutsch suggested that I make the listings available to the rest of the local user community. So around February 1990 we found the space, placed all the listings there and announced the information.

Around April or May 1990 Peter read an article on one of the newsgroups with somebody asking the generic "Where can I find this software?" question. He did a search (*grep*(1)) through the files (which were just stored as raw directory listings) and replied to the net with the answer to the question. He then received a number of requests either along the line of "Can you locate this for me?" or "Can you make this facility generally available?". The initial response on our part was to make the listings themselves available for anonymous ftp. Just before I was about to leave for the USENIX conference this summer, we co-opted Bill Heelan (wheelan@cs.mcgill.ca) to write a simple user interface to the listings. This was called **listd** and operated on the same basic principle as archie (you opened a telnet to `quiche.cs.mcgill.ca`, logged into **listd** and issued the **prog** request). However listd just performed an *egrep*(1) on the raw listings and returned the matched lines. All in all, it was rather simple, but we soon were getting about 30 logins a day.

While I was at USENIX, Peter received a letter from Jerry Peek (who was then at Syracuse university, now at O'Reilly Associates, jerry@ora.com) who had in place a (much more ambitious) set of shell scripts to do a similar thing, as well as anonymous ftp to allow the net to access the listings. I met Jerry at the conference and we discussed the concept of setting up a dedicated database. When I returned to Montreal, Peter and I and worked out a preliminary structure for the database which I subsequently modified as the requirements for the project changed. The initial version of the what later was to become known as archie was released towards the end of November 1990.

Subsequently, archie has received widespread and enthusiastic use from the Internet community. From usage that averaged about 30 logins per day originally, archie was servicing over 770 logins/day by the beginning of April 1991. The email interface to archie was announced on December 18 1990 and has seen a similar explosive increase in its use. See Appendix 2. An article on archie appeared in the February 1991 edition of the *SunTech Journal*, a UNIX trade magazine.

At the time of writing a number of cooperative ventures are being worked out in order to have archie distributed to other sites around the world. Various organisations have expressed interest in running an archie service and an archie protocol will soon be defined to allow a client/server model to be implemented.

Appendix 2
Archie Usage

Archie usage has increased by over 400% since it was first introduced to the Internet and had a total usage of 45,217 logins during the period December 1 1990 and March 31 1991. The following graphs plot the number logins/day of the interactiveand email interfaces during this general period. Dates marked on the graph are Sundays of the month.

Appendix 3
Archie Statistics

The following pages contain miscellaneous information gleaned from the archie database.

We have confirmed usage of **archie** by the following countries:

.au	Australia
.at	Austria
.be	Belgium
.br	Brazil
.ca	Canada
.ch	Switzerland
.de	Germany
.dk	Denmark
.es	Spain
.fi	Finland
.fr	France
.gr	Greece
.il	Israel
.in	India
.is	Iceland
.it	Italy
.jp	Japan
.kr	Korea
.my	Malaysia
.nl	Netherlands
.no	Norway
.nz	New Zealand
.se	Sweden
.su	Soviet Union
.tw	Taiwan
.uk	United Kingdom
.us	United States

As of April 12 1990 **archie** was keeping track of **44,697,686,988** bytes of archived data stored in **981,497** files.

Monthly usage totals (interactive interface)

December	5250
January	8004
February	11279
March	20684
Total	45217

Top 20 sites (by size):

Host	Files	Bytes
gatekeeper.dec.com	25953	1,391,515,104
wuarchive.wustl.edu	31000	1342527977
ux1.cso.uiuc.edu	28531	1199379272
mrcnext.cso.uiuc.edu	12205	1076101173
psuvax1.cs.psu.edu	50893	805016039
uunet.uu.net	24127	734539525
forwiss.uni-passau.de	14241	611631446
mcsun.eu.net	9042	601301914
hp4nl.nluug.nl	6652	597001274
tupac-amaru.informatik.rwth-aachen.de	13545	587051113
metro.ucc.su.oz.au	11855	573356750
derro.ucc.su.oz.au	11794	571008954
apple.com	14753	543925804
walhalla.informatik.uni-dortmund.de	7655	527277037
sachiko.acc.stolaf.edu	20722	479126202
sol.deakin.oz.au	8257	463552312
aixpsb.rrz.uni-koeln.de	3485	446698229
ugle.unit.no	7140	444252213
utsun.s.u-tokyo.ac.jp	7900	415130859
cicero.cs.umass.edu	2397	409728219

Appendix 4
Source Listings